1-1-2006

An integrated task manager for virtual command and control

Thomas J. Batkiewicz
Iowa State University

An integrated task manager for virtual command and control

by

Thomas J. Batkiewicz

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Human Computer Interaction

Program of Study Committee:
Adrian Sannier (Major Professor)
James H. Oliver
Eliot H. Winer

Iowa State University

Ames, Iowa

2006

Graduate College

Iowa State University

This is to certify that the master's thesis of

Thomas J. Batkiewicz

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The Task Manager is a desktop/tablet PC interface to the Battlespace research project that provides interactions and displays for supervisory control of unmanned aerial vehicles. Utilizing a north-up map display, the Task Manager provides a direct-manipulation interface to the units involved in an engagement. Used in two primary modes, the Task Manager can be used either in a planning/review mode that can be used to generate mission scenarios or a live-streaming mode that connects to a live Battlespace simulation via a network connection to edit and update path information on the fly.

The goal of this research is to combine the precision of 2D mouse and pen-based interaction with the increased situational awareness provided by 3D battlefield visualizations like the Battlespace application. Combined use of these interfaces, either by a single operator or a small team of operators with task-specific roles, is proposed to produce a more favorable ratio of operators to units in field operations with superior decision-making capabilities due to the specific nature of the interfaces.

CHAPTER 1. OVERVIEW

Currently, there is widespread use of unmanned aerial vehicles (UAVs) across all of the branches of the U.S. military. UAVs provide a unique capability in their nature as completely mechanical units, meaning that they can specialize in tasks unsuited for humans. Exposure to biological contaminants, extremely high-risk combat missions, and intelligence, surveillance, and reconnaissance (ISR) missions are just some of the tasks where the UAVs are exploited to great advantage, leading to the convention that UAVs are well suited for tasks that are categorized as "the dull, the dirty, and the dangerous." As such, the Department of Defense projects increases in the use of unmanned aerial vehicles in military operations [1]. As the capabilities of UAVs grow, so will the range of tasks they will be expected to perform. The increasing complexity of their behaviors, coupled with the dynamic role they will play in the future of military operations provides a research challenge for interface design. Additionally, there are other possible uses in commercial transport, homeland security, and others too numerous to expound upon which may motivate this research.

1.1 Introduction

In order to meet projected demand, the current structure of the interface between the operator and the vehicle must undergo a serious change. The Department of Defense's Unmanned Aerial Vehicles Roadmap 2005-2003 calls for an inversion of the pilot to vehicle ratio; currently multiple people are required to operate a single vehicle [1]. Future advances in the automation of these units will enable this fundamental shift in interaction, but the interface between the operator and the units must account for a corresponding increase in the amount of information available to the operator. Switching from a paradigm of many-to-one to one-to-many runs the risk of consuming the operator's attention entirely. Overloading the

operator's attention will only decrease their situational awareness and impair their ability to make the correct decisions.

This will necessitate a change in the role of the operator of the UAV. No longer will they serve solely as the pilot, but as the manager and mission leader. Current UAV interfaces rely on a stick-and-rudder combined with a map display and video feed to directly pilot the vehicle, albeit remotely. Piloting with this camera-limited view has been compared to trying to fly while looking through a soda straw. The older "soda-straw" methods for UAV control will not be adequate for the management of multiple UAVs; constant micromanagement of individual units will quickly overwhelm an operator. This operator is solely responsible for flying the aircraft, and a separate operator is responsible for managing the payload of the vehicle, be it sensors or munitions. An example interface can be seen in Figure 1.

Figure 1. Example UAV Interface

In the next generation multi-unit paradigm a wider perspective will be required for the operator, providing them with information on all the UAVs in their engagement theater,

as well as other information from outside sources [2-4]. Furthermore, the ability to direct UAV swarms; providing high-level commands to large numbers of UAVs will be of use when controlling the UAVs of the future [5]. Operators need a high-level portal to the engagement as a whole to provide them with the situational awareness to properly utilize the UAVs they control [2-4]. Fusing all available information; from terrain and weather data, unit positions and movements, radar tracks, and video feeds into a single cohesive display will enable this future manager to assume their role, which is more similar to command and control than piloting. The ability for an operator to get detailed, relevant information will assist them in making quick, effective decisions, and the ability to drill down to the level of the pilot or tele-operator will still be required in future interfaces [6].

1.2 Background

The high-tech battlefield of the future is already emerging – it will eliminate paper mapping with grease pencils, and instead rely on digital datalinks between the commanders and the soldiers and units in the field. However, visualizing an entire battlefield in an effective manner is a complex, but significant task [7]. Information fusion, presentation, and the support of real-time decision making are key requirements for this type of visualization, and challenge existing interface design conventions. Using 3D visualizations to increase situational awareness is not a new concept, and although it has been successfully tested, it has not seen adoption into mainstream use.

The Dragon project is one such experimental battlefield visualization platform designed to maintain a high level of situational awareness [8]. Using 3D graphics projected by a digital workbench, a top-down view of the battlefield is presented, with terrain information, unit entities and symbols, and operations plans [9]. Using multiple interaction modes, users navigate this environment and gather information about the engagement. A screenshot of this application is shown in Figure 2.

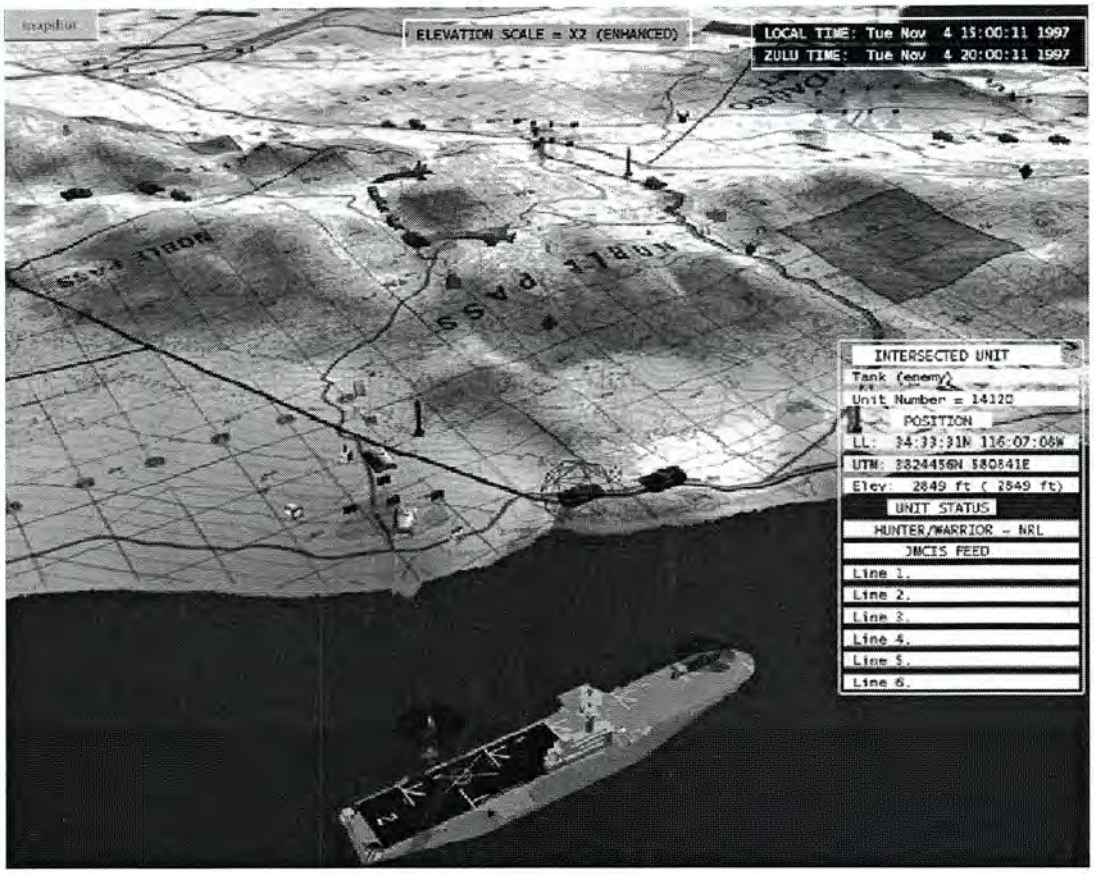

Figure 2. Dragon Visualization [9]

The Dragon visualization utilized multi-modal interaction, and focused not on map-based navigation but instead gave the user control to pick and choose a viewpoint which supported their ability to gain insight into the situations of specific units in the scene [9]. Also, this project leveraged existing military symbology combined with realistic models to represent the various units in the engagement [8]. Borrowing Dragon's visualization techniques, such as unit aggregation, serve as a preliminary benchmark for 3D battlefield visualizations.

Falconview is another battlefield interface application, but one that has been adopted for many uses across several branches of the U.S. military. Falconview is a moving-map display that can render terrain data and other information overlays on a desktop application

[10]. Originally developed as mission planning software for F16 aviators, it has been expanded to other uses in geographic information, including UAV control as well as onboard navigation of some aircraft. Currently, it is part of the Portable Flight Planning Software (PFPS) package and its mapping engine has been chosen as the standard for the upcoming Joint Mission Planning System (JMPS) [10]. Leveraging existing applications in an effort to better span the gap between 2D interfaces and 3D visualizations will be crucial in successful deployment of these technologies.

Current UAV interfaces are extensively tested for the U.S. military before they are used in field operations. The Air Force Research Lab has developed experiments for investigating human factors in UAV control and a testbed system for the interface between operator and unit [11]. Work has been done on the anticipation of an increase in the level of automation of UAVs, as well as supporting two different control styles: manage-by-consent and manage-by-exception [11, 12]. In both modes, the UAV is assumed to be able to dynamically re-plan its mission, automatically generating alternate paths and identification of targets [12]. In manage-by-consent, the UAV will not take any action upon these new paths unless confirmed by the operator, and in manage-by-exception these adjustments are made automatically after a specific time period unless the operator specifically objects [11, 12]. This testbed utilized a map display showing planned paths, threat areas, targets, and other aircraft on one monitor, and the other shows images captured by the UAV and sent to the operator for automatic target recognition (ATR) verification, with keyboard and mouse inputs [12]. An image of the interface setup is shown in Figure 3.

Figure 3. Multi-UAV Task Environment [12]

Test results showed that although manage-by-exception yielded higher levels of automation, it actually put the operators under more stress, since there was a time constraint on their decisions [12]. Manage-by-consent ensured that if the operator was unable to respond to a certain unit's planning, the unit continued on the operator's last-known plan for that unit. In manage-by-exception, if the operator was distracted with a high workload for a short period of time, many of the units would have automatically changed their plan and lowered the operator's situational awareness [12].

Other work on situational awareness has demonstrated a desire and a need for a map-based display for various information overlays and unit positions [13]. The importance of a cohesive interface to minimize issues with window management was also expressed to reduce workload and operator error [13]. One type of application that typically conforms to

this type of interaction style, and also addresses real-time decision making is video games. Video games have been used as training tools in various capacities for the U.S. military, as well as other research into many other domains [14]. The use of commercial-off-the-shelf technologies for video gaming can provide satisfactory alternatives to keyboard/mouse interaction, since the free market typically rids itself of products that are difficult to use [13, 14]. Examing interaction methods for 3D environments in video games can be a source of inspiration and effective techniques [14].

Additionally, branching out further into multi-modal interaction for the command and control of UAVs could allow for a natural language interface [15]. The level of automation UAVs provide could be accessed by issuing voice commands to the UAVs, much like a current commander does with their human troops deployed in the field. Building a task information system that can respond to this type of information could be combined with existing 3D visualizations or 2D map interfaces. One other option for inverting the ratio of pilots to vehicles is to scale up both factors to find an equitable balance. Team-based interaction could allow multiple operators with distinct roles to share responsibilities and increase their total situational awareness [16].

1.3 Research Goals

From this background, the goals of this thesis research were formed. First, a common framework must be created to aid in the development of software applications for battlefield visualization. A second goal is to develop interfaces capable of presenting battlefield visualizations that provide a high level of situational awareness with varying levels of detail, combined with precise control mechanisms that allow supervisory command and control. This thesis presents research done on these objectives towards a potential future interface suite. Existing research was examined to learn the lessons of others who have been working

on this interface modality, and to find comparative advantages and disadvantages of different approaches to this unique problem set.

CHAPTER 2. SYSTEM ARCHITECTURE

This thesis presents a software application that is a potential interface for the supervisory control of multiple UAVs. This application relies on a foundation of common software libraries. These libraries, coded in C++, create a basic framework that allows for rapid development of the various components and modules needed in UAV control applications. The overall architecture follows the model of object-oriented programming, which allows for modular, scalable design. Subcomponents often derive from a set of basic objects that are used extensively throughout the code.

Further, the architecture employs several open-source libraries that aid in the prototyping required in interface design. Use of these software foundations allows researchers to focus on the critical portions of the interface design and implementation, rather than spending their time building the entire infrastructure needed to support the application. The open-source libraries expose the underlying code so that when necessary it can be modified to fit specific requirements. Existing design patterns were used to derive this architecture [17].

2.1 Components

This research platform makes extensive use of object-oriented programming concepts. Objects serve a clear, unique purpose and encapsulate a very specific behavior. The goal was to create a simple, flexible architecture that minimizes object dependence and enables scalable interactions. Composed of only three main components, this architecture implements a basic framework for inter-object communication that is easily extended to communicate between objects on remote computers or in different applications.

The first component is the command object which is the message object passed between objects without requiring direct knowledge of what objects are loaded. The other components form a two-layer hierarchy, with the top layer being composed of a single object.

This top layer is responsible for managing the second layer and synchronizing communication between components on the second layer. The top layer has a pointer to each member on the second layer, but only sees them as instances of a base class that implements the communication protocol. The objects on the second layer all derive from a base class, and are decoupled from one another; no pointers are exchanged among the second layer. The communication between these components is achieved by upwards propagation of the command-class messages to the top layer, where they are redistributed to all the components on the second layer.

2.1.1 Command class

The command object is central to the distributed nature of the system's architecture. Communication between modules and even networked applications relies upon this class. Each instance of a command encapsulates one instruction or update, and it contains all the data relevant to execute that instruction or interpret the update. The command's simple structure allows it considerable flexibility in expressing information that must be shared between components. The minimum size for each of these instructions is 16 bytes, which makes it suitable for both intra- and inter-application communication.

Commands include two top-level pieces of information, the directive and the qualifier, which together serve as the message header, and indicate the rest of the command's contents. The directive typically specifies what has changed in the application, and the qualifier is used to provide additional context or designation. These two fields are represented in the code as an integer that draws its value from a globally defined enumeration; a collection of named integer values. This simple underpinning keeps the size of the command structure small; 4 bytes each for the qualifier and the directive, while preserving the ability for user/programmer readability. Further, since the enumeration is a

specific type, at compile-time all assignments of the enumeration are validated which can reduce time spent debugging.

Additionally, the command structure contains two dynamically-sized arrays for passing additional types of data – character data and numeric data. Character data is handled in the form of adding strings to the command. Numeric data is held in floating-point form, but integers and Boolean values can be cast into this type as well. In this manner, we are able to pass the common C++ types in a concise format.

2.1.2 BattlespaceComponent class

In the Battlespace applications, the components on the second-highest level are known as managers. Each manager encapsulates a specific functionality such as handling input devices, managing a scene graph, or updating vehicle dynamics. The managers are instances of C++ classes that can be dynamically loaded or unloaded by the application as needed. To allow communication between managers, each manager shares a common set of functionality by deriving from a base class, BattlespaceComponent. The BattlespaceComponent base class provides a common interface that is used to share and synchronize updates between the managers, as well as providing a framework of handling the inputs from other managers. Each manager can register its own member functions to be automatically called when a command with a specific directive/qualifier is received through this architecture.

This simplifies the amount of work that must be done when creating new components or modifying existing components. Interdependence between objects is minimized, since pointers to other components on the same layer are not necessary. Effectively, the managers are able to make function calls on other managers via this command communication, without hard-coded linkage between components.

2.1.3 ComponentManager class

A higher-level manager is responsible for synchronizing the communication between the other managers. This module, the ComponentManager, is similar to the event-processing system used in most modern GUI libraries. It is responsible for polling individual managers for their updates and disseminating this information to all the other managers. The functional cycle in which this process takes place is called the command loop. In each iteration, the ComponentManager calls the updateState virtual function implemented in each manager and inherited through the BattlespaceComponent. This function passes back a list of commands that this manager has created since the last iteration. Once it has completed this process for each subcomponent, the command list is the collection of changes since the last iteration and provides all of the information required to bring other modules into synch.

Once it has assembled a complete list of commands from each manager, the processCommands function is called on each manager. The managers can then react to these commands, or ignore them, based on their specific functionality. In this fashion, the managers are not required to know which other managers are loaded; they simply need to be able to process the commands related to their function and put out relevant updates for other managers.

2.1.4 Networking

For the purposes of our research, the main Battlespace application must run on a distributed cluster of machines. The use of the command structure as the only mode of communication between managers greatly simplifies implementation on a distributed memory architecture as well as networking with other applications. After the command loop collects all the commands from the managers, it must simply share this list with each node in a cluster at a specific time to synchronize it. Since the command structure is so simple and is constructed out of basic types, serializing this structure to be sent across a network is trivial.

Furthermore, the process to network in other applications is fairly simple. All that is required is to create a new manager type, the NetworkManager, to encapsulate this behavior. Since the command structure is already serializable, the NetworkManager merely sends and receives commands via socket communication. This manager doesn't generate its own commands, nor does it process them; it simply makes the internal command list available to outside applications, and upon receiving commands from external sources adds them to the list to be processed internally.

This makes it possible for a component on one manager to send commands to a remote component as transparently as it would be if the component were co-located within the same application. The flow of commands between networked applications is shown in Figure 4.

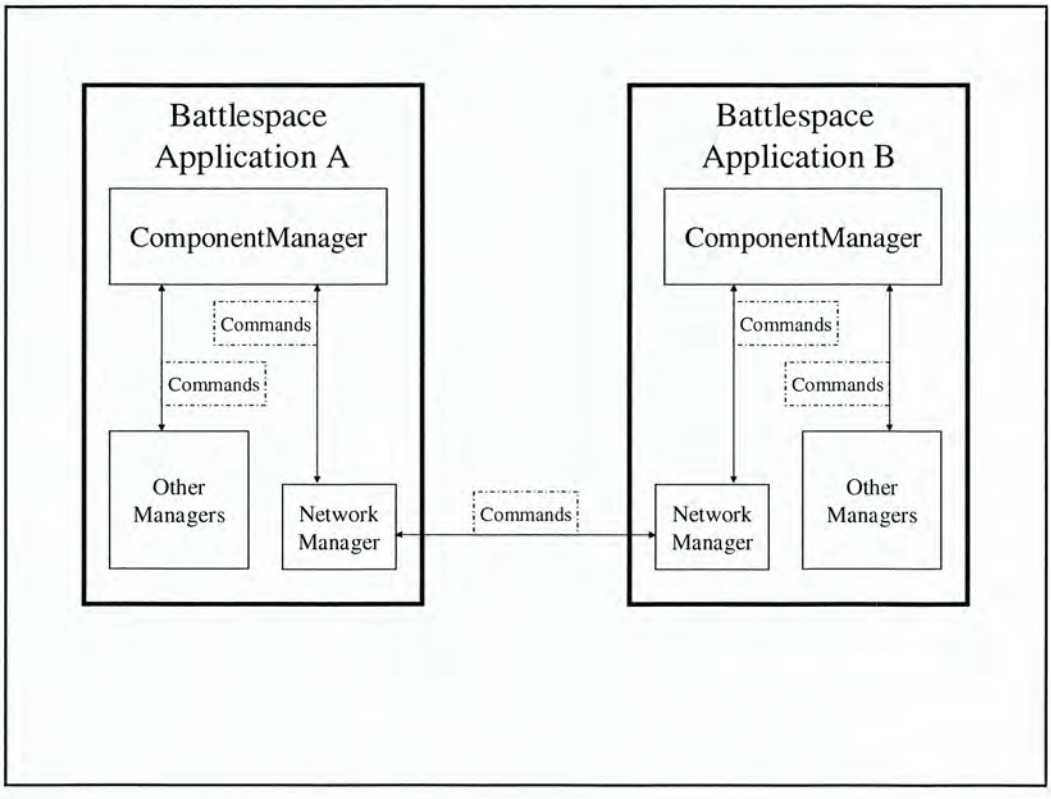

Figure 4. Networking Diagram

CHAPTER 3. BATTLESPACE – 3D VISUALIZATION

The Battlespace application is a real-time 3D graphics application that provides dynamic viewing and interaction with units distributed over a large battlefield. Battlespace is built upon the premise of using a primarily synthetic environment to build a baseline 3D visualization and integrating live sensor feeds directly into this environment, constantly updating and inserting the most up-to-date information into a fused display. The goal of this approach is to maximize the user's situational awareness by combining what would typically be several applications into one. Further, the Battlespace application supports a wide range of interaction schemes, from controlling a swarm of hundreds of UAVs with general behaviors, supervisory control of a squadron of UAVs, to direct teleoperation of a single UAV. Supporting this large array of interaction and display modalities in a way that does not overload the user's attention is the chief research problem.

3.1 Open-source Components

Battlespace is built upon several open-source packages that allow the research to be focused on the display interfaces and interaction techniques rather than programming basic software libraries.

3.1.1 VRJuggler

VRJuggler is a portable C++ library suite that offers a virtual platform for development of virtual reality applications [18]. Abstracting the devices from the application, VRJuggler enables the developer to "code once, experience everywhere," [18]. Display and input device differences are handled at run-time with the use of an XML-based configuration system, allowing the same binary executable to drive a simulation on a desktop, a power wall, or a 6-sided immersive cave.

3.1.2 OpenSG

OpenSG is a cross-platform, scene graph package used by the Battlespace application for rendering [19]. Built on top of the OpenGL graphics programming language, it provides real-time 3D graphics. OpenSG also supports multi-threaded access to the scene graph data, and provides a unique distributed approach to scene graphs which can help improve performance across a cluster [19].

3.1.3 Demeter

Demeter is a cross-platform terrain engine that supports the loading and rendering of geographical data [20]. Based in OpenGL and built in C++, Demeter uses adaptive rendering techniques to maximize performance while maintaining a high visual quality [20]. Demeter supports multi-texturing which is a key feature in Battlespace as overlaying information directly onto the terrain itself is a valuable display method. Further, Demeter is written to be a self-contained object which makes for simple integration into Battlespace.

3.2 Battlespace Application

Battlespace operates in two primary modes; a strategic mode providing a "gods-eye view" with symbolic representations of units, and a pilot mode that enables visualization of units in a realistic scale and provides a chase-cam view. Changing between the two is a seamless, context-preserving, view-interpolating transition that allows the user to maintain a high level of situational awareness. All of the various interaction devices can be used in both modes to adjust the visualization and direct the units. In general, the operator is unrestrained in their ability to view the battlefield, allowing the operator to decide what areas should receive their attention. However, there are times when emergency situations arise that require an operator's immediate attention. The chief way that the operator's attention is called to areas requiring their input is through the alert system present in Battlespace.

The alert system is a graphical notification system that can be triggered by a number of different issues such as low fuel, threat/target proximity, etc.... Any time an entity enters into a situation that requires an operator's attention, a visual notice appears on the display. One button press will take the operator to the new alert and present them with the details of the problem, as well as the information required to make a decision.

3.2.1 Strategic Mode

In strategic mode, the application displays the battlefield from a gods-eye view, providing a high-level view of unit's positions and goal states. Typically, the viewpoint is set several miles above the earth's surface to provide full coverage of the engagement. The operator can manipulate the viewpoint to any location and orientation, allowing them to choose what to see. This presentation is aimed to maximize the situational awareness of the user, but since it relies on a 3D visualization there are some key factors that must be taken into account so that the information is not ambiguous.

3.2.1.1 Visualization Enhancements

One of the primary ways in which mission commanders view engagements is from 2D, top-down displays that overlay altitude information onto them. However, from an arbitrary viewpoint in a 3D visualization, this information may be misleading or deceptive, causing the operator to misunderstand the situation. Since the ground position and altitude can be difficult to interpret with the visualization of a vehicle in midair, we developed the height stick to compensate for this. With alternating bands of red and white, this stick extends from the bottom of the unit down to the ground. Each band represents 5000 feet, and so a rough measure of altitude can quickly be determined at a glance. Further, to identify ground position, a small cylinder was placed at the base of the height stick to help the operator identify the ground position directly below the air vehicle. A view of several units with height sticks enabled is shown in Figure 5.

Figure 5. Vehicle Height Sticks

Another visual feature that helps the operator maintain situational awareness is the history trails. History trails are used to show the path each vehicle or set of vehicles has followed. A colored path is overlaid directly onto the terrain rendering which denotes the past positions of the entities. The colors are used only to indicate whether the trail leads to friendly (blue) or hostile (red) forces. This allows an operator who has spent time examining one section of an engagement in detail to quickly catch-up to the current situation.

For unmanned entities, the future flight path may also be displayed in the Battlespace. Since the altitude of the waypoints along the path matter, this must be done in 3D. Similar to the method for height-sticks, the future flight paths are shown with alternating bands that correspond to 5000 feet of elevation. Waypoints are drawn as vertical columns along the path, creating a visual structure that resembles a fence. This path visualization is shown in Figure 6.

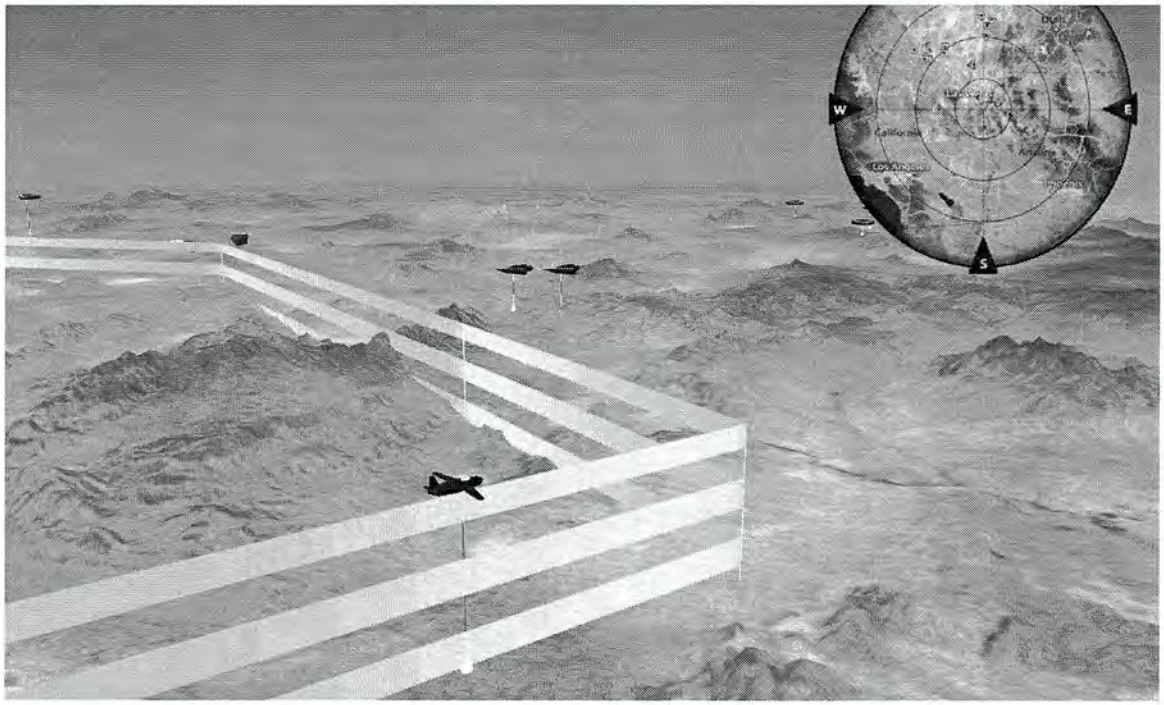

Figure 6. Future Flight Path Visualization

Users can also utilize a heads-up display that provides a north-up 2D radar view. It shows current unit positions and orientations in a birds-eye view, and uses a fixed-position reference for the user's position, with a semi-transparent white overlay to indicate the primary viewing frustum (e.g., the front wall of an immersive CAVE display). All this

information is layered upon a map that provides terrain/political data for the immediate area. A close-up view of the HUD Radar is shown in Figure 7.

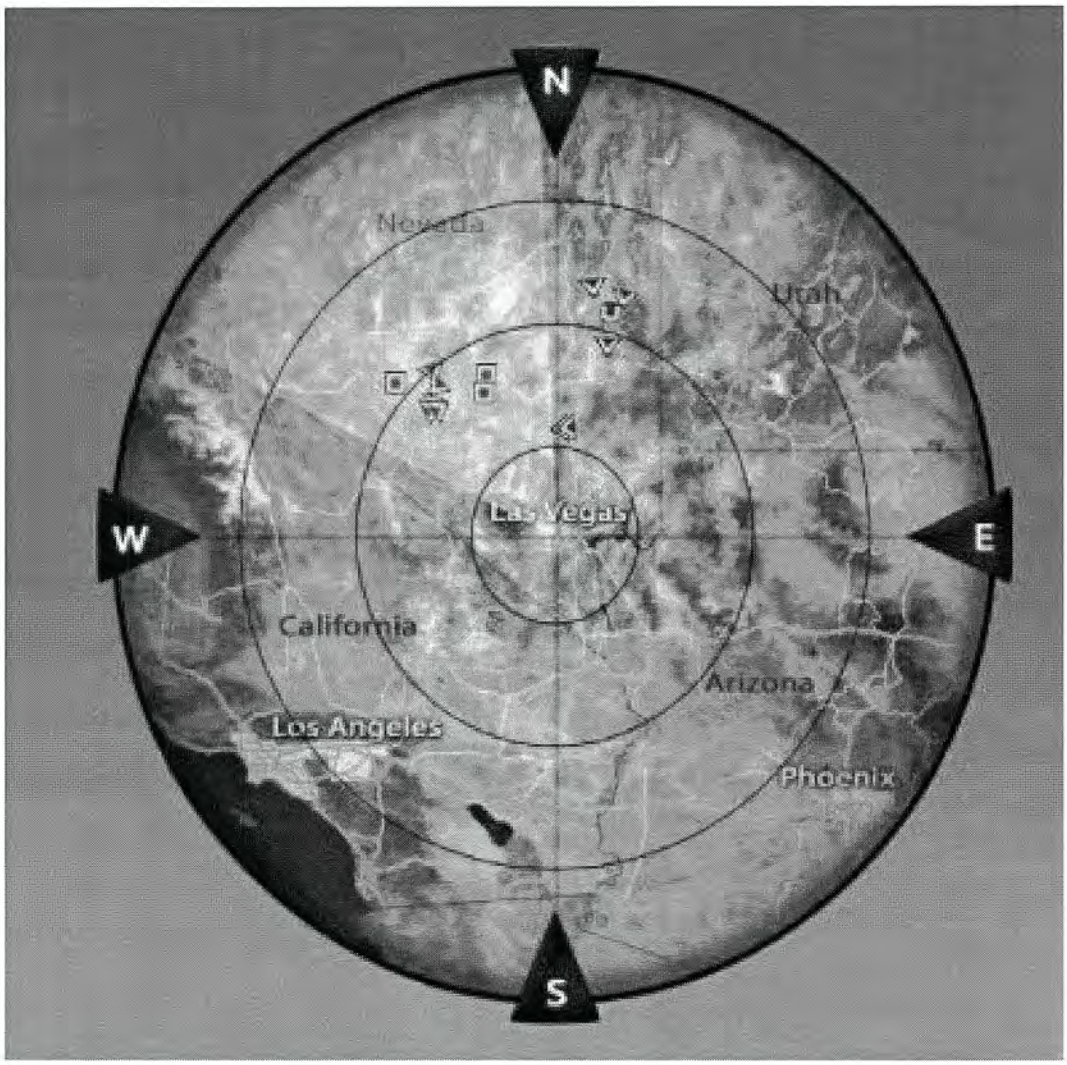

Figure 7. HUD Radar

3.2.1.2 Unit Information

Another key feature of the Battlespace application is its approach to displaying the properties and attributes of units in the engagement to the operator. The operator's

knowledge of the specific capabilities of their units, as well as those of hostile forces, is important to maximize combat effectiveness. Visualizing unit-specific information rather than information regarding the unit's position, plan, or the general flow of battle is important to providing the necessary details required for command and control.

The ability for an unmanned unit to act independently of an operator is strongly based in the capabilities of its sensors. Sensors onboard unmanned vehicles, as well as weapons systems may have maximum ranges to which they are effective. The ability for an operator to understand what the UAV is able to "see" will be an important factor in command and control, especially as the level of autonomy grows. Furthermore, the display of weapons capabilities for both friendly and hostile units can be helpful for an operator. In Figure 8, this display of threat and sensor information is shown: the hemispheres of overlapping concentric red rings indicate the airspace threatened by a group of surface-to-air missile launchers, and the green wedge indicates a forward-looking sensor on the lead incoming unit.

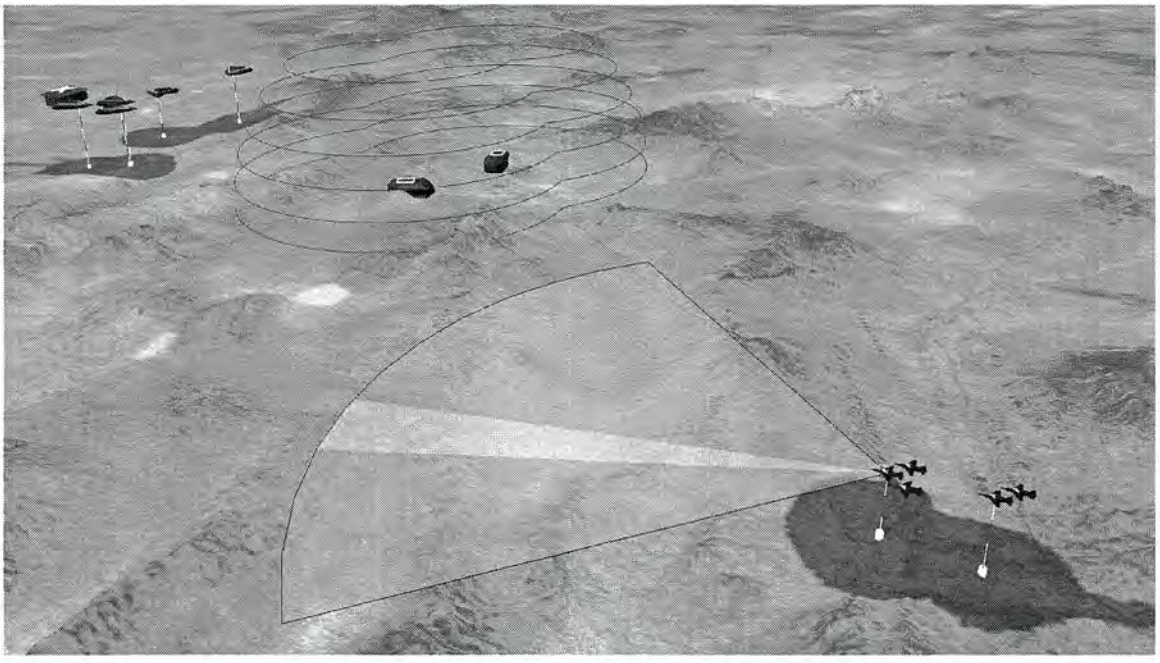

Figure 8. Sensor & Threat Displays

Another primary source of information from an unmanned vehicle is the cameras mounted on it. Relaying the picture back to the operator is standard practice in current pilot-oriented interfaces, and should remain integrated in future interfaces. One of the ways this information is handled in Battlespace is to register sensor imagery to a location and provide a specific symbol for it. If the image sent back by the UAV is noteworthy, perhaps designated as a potential threat by automatic target recognition (ATR) or from some other source, the operator will need to know this. Using the alert system previously mentioned an operator's attention can be called to view this new development. If the ATR was unable to make a determination, the operator can examine the video information and classify it, as shown below in Figure 9.

Figure 9. Alert Classification

This approach is suitable for situations in which the UAV has already flown past this location and gone on to other tasks, but real-time streaming camera information is handled differently. Placing the camera feed in context in this 3D environment would be an extension of streaming video's use in current unmanned ground control systems, augmenting the soda-straw view with our synthetic terrain. However, this information is specific to the viewpoint of the unit, and will be handled in the other main mode of Battlespace, the pilot mode.

3.2.2 Pilot Mode

Pilot mode is the second operational mode for the Battlespace application. In this mode, vehicles are rendered at a realistic scale, and generally the operator's viewpoint is localized to a specific unit, with the default being a chase-plane view of the currently selected unit. Other fixed viewpoints are provided that give a variable distance to the unit, as well as providing left and right-looking views. Since the vehicles in pilot mode are rendered at a realistic scale, vehicles not close to the viewpoint are difficult to see due to distance, as in real life. A colored (red/blue) hemisphere is used to indicate position of such remote units, and the height sticks described in the strategic mode section are also used. These features are shown in Figure 10 which depicts a typical view in pilot mode.

Figure 10. Pilot Mode

Also in pilot mode, precise values can be provided for specific vehicle information such as altitude, speed, heading, etc. All the things that would typically be included in a pilot's HUD can be displayed in pilot mode. Further, camera information can be added in context, so that its orientation relative to the vehicle can be replicated in the simulated environment, overlaying the synthetic information with the most current information available. An example of a pilot mode HUD is shown in Figure 11.

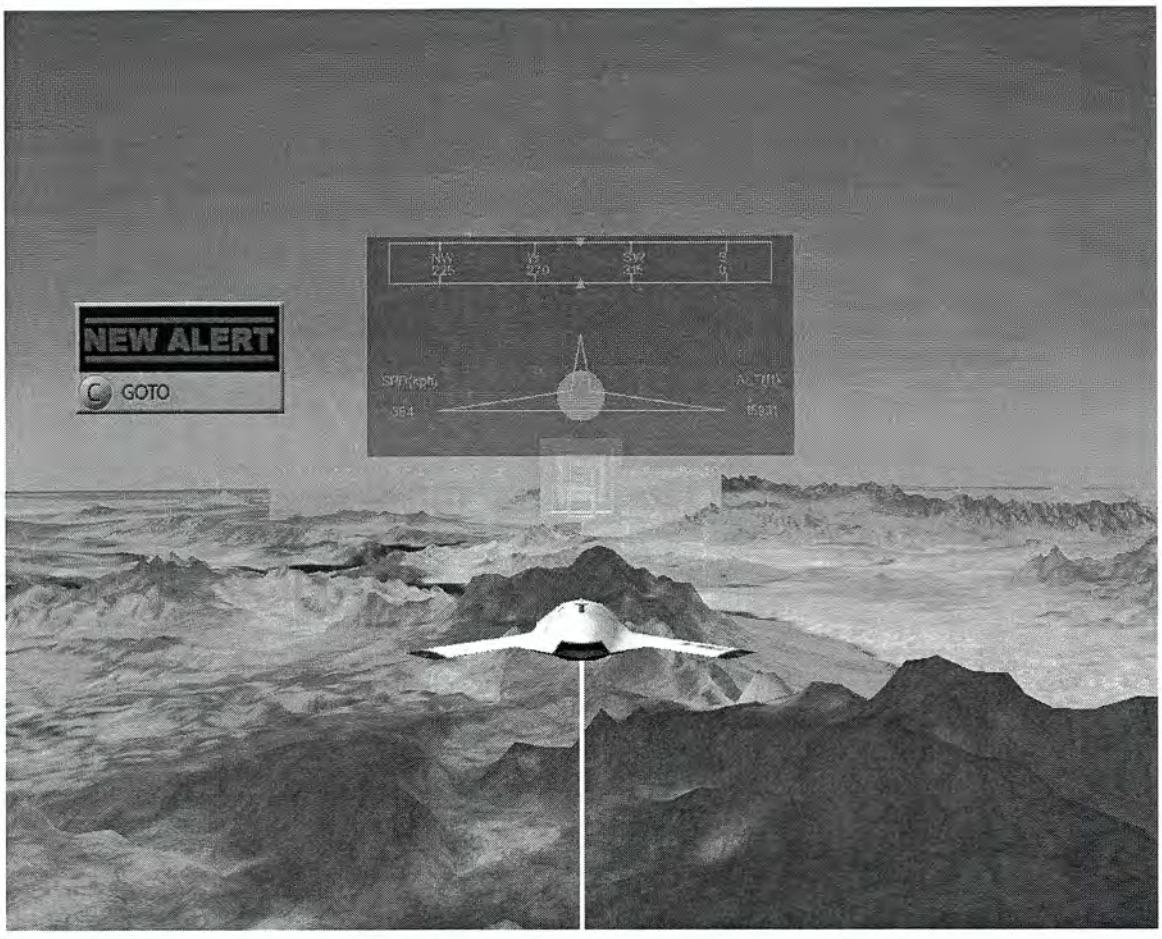

Figure 11. Pilot Mode view with HUD

Another capability that pilot mode deals with is the issue of 3D path generation, visualization, and planning. When a unit is approaching a threat, the operator may wish to choose a new path that minimizes exposure, maximizes speed, or follows some criteria to achieve an objective. An algorithm can be run to generate alternate paths, but ideally the operator should be allowed to decide which path best achieves the objectives. Figure 12 shows the display for an operator choosing between alternate paths. Upon selection the path is updated and the unit will follow its new path [21].

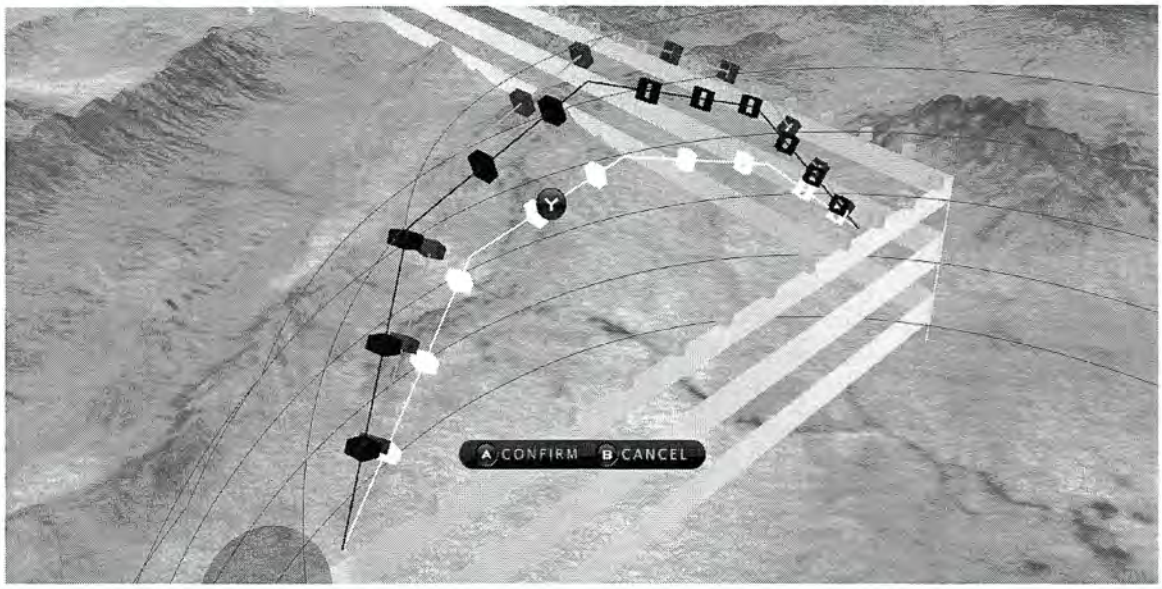

Figure 12. 3D Path Editing

Lastly, pilot mode provides the displays necessary for teleoperation. With an egocentric viewpoint with respect to the unit, a pilot can have all the sensors and instrumentation displays integrated into this synthetic environment. Rather than having various instrumentation displays scattered in front of them, a single fused display can provide them the data required to fly the aircraft remotely. Basing it on the virtual environment allows for the commander to take control, whether directly or indirectly, of the unmanned vehicle as the situation demands.

3.2.3 Display Devices

A variety of display modalities are developed in the course of this research. Starting with a basic desktop computer, the Battlespace is used in a day-to-day fashion on standard CRT or LCD screens as a part of a development platform. However, since our aim is to increase situational awareness, we utilize several immersive displays that fill the user's field of view. Such displays typically require less view manipulation since the user is free to simply turn their head to view more information.

The primary immersive display for the Battlespace is the C6, a 10x10' room where all six walls are rear-projected screens. This allows for the user's entire visual field to be filled, and no matter where they turn all they see is the virtual environment. Though equipped for active stereo, this project runs in monoscopic mode since the distances are so great as to make stereo rendering unnecessary. Battlespace has also been run on the C4, which is a dual-configuration system similar to the C6, and the Babycave. Both provide 3 back-lit screens in front of the user, and the C4 provides a front-lit floor in addition.

The remaining displays fall into the category of head-mounted displays. Some experimentation has been done with a pair of virtual binoculars, which when tracked with respect to the screens provide a user with the ability to view the engagement at a magnified scale. Though only preliminary, this work will likely be continued to develop role-specific interfaces that expand upon the differentiation between strategic and pilot mode. Ideally when completed, the research would define roles for a small team of operators and custom displays and interactions for each which benefit from and are integrated with the large scale immersive display.

3.2.4 Interaction Devices

Input to the Battlespace application is handled in a number of different ways. Though keyboard/mouse interactions would be appropriate in a desktop setting, they require a surface to operate upon which would limit the user's field of view. Defining a standard set of input devices will help us measure the effectiveness of display. With that in mind, we set out to find some interaction devices that would utilize the input capacity of both hands simultaneously but do not have a large physical footprint, and would allow for rapid input for time-critical decision making. Not surprisingly, our first input device is borrowed from another real-time decision making domain; video games.

3.2.4.1 Gamepad Controller

The gamepad input device is a wireless Logitech game controller that has 2 analog joysticks, an analog sliding throttle control, a digital direction-pad, 7 usable primary buttons and 4 shoulder buttons. Two of the buttons on the controller are unusable by the application since they are used to toggle hardware features on the controller only. An image of the controller is shown in Figure 13.

Figure 13. Gamepad Controller

One of the key features of the gamepad interface is that although there are only 11 usable buttons, there are a far greater number of interactions possible based on the implementation of the interface. Internally, the application maintains a keyed table of button signals to function pointers. These function pointers can be reassigned upon receiving commands from within the Battlespace and also by the functions called when a button is pressed. For example, this means that the 'A' button can be used to trigger selection mode, but upon selection, the 'A' button can then be used within this selection mode. Buttons can be remapped on the fly to achieve desired interactions.

This feature keeps the buttons closest to the resting position of the user's hand available for constant input which makes for quicker interactions. This also allows the buttons at the fringe of the device to be saved for specific functions that don't change over time, such as responding to alerts (which may decrease reaction time since it is always the same button). Further, development speed can be increased using this framework. Developers can simply define high-level operating modes and write functions that bind the correct functions pointers to button presses. Any function called by a button press can rely on calling these mode-setting functions to easily transition between interaction schemes, without having to be directly involved with the development of them.

Though the controller interface has these benefits, in itself the controller provides no visual feedback to the user. Requiring the user to memorize all the possible use modes is not a viable strategy, so other components must be integrated to provide a meaningful interaction scheme. A menu system provides a visual listing of actions the user could take and is a fairly well-developed scheme that finds use in everyday applications.

3.2.4.2 Menu System

Menu systems are very common and simple interaction methods, since menus provide a visual reference and confirmation of your actions. In Battlespace, the menu is rendered in 3D as a billboard, so that it always faces the user. From the menu, users can toggle visualization features on and off, pause or slow a simulation replay, and view/edit targeting information using the controller. These various categories are grouped into separate panels in the menu. For example, the visual features panel is shown below in Figure 14.

Figure 14. Visual Features Menu

Input to the menu is done through the gamepad controller, and each menu provides a visual key to for use of the buttons that are related to that menu. The shoulder buttons on the controller are used to switch between menu pages, and the directional pad is used for selection within a single page.

3.2.4.3 Voice Input

Another input device that is easily integrated via wireless networking is the input of voice commands. A wireless headset microphone can be worn by the user and, using a speech-to-text engine, the user's spoken words can be parsed and translated into commands. One of the most useful features of the voice input system is the ability for users to 'tag' a unit

with a particular designation. Units can be given specific names by the user that can then be understood by the system when combined with existing commands. For example, a unit designated 'alpha', could then be selected when the user says 'select unit alpha'. This type of interaction reinforces the idea that Battlespace provides a level of control greater than that of a typical pilot and more on the level of a commander.

CHAPTER 4. TASK MANAGER – 2D INTERFACE

The Task Manager is a desktop application that utilizes a birds-eye view, north-up map display of the battlefield. It provides a direct manipulation interface to the entities on the battlefield. Work has been done on both a traditional desktop with keyboard/mouse interaction as well as a wireless tablet pc with pen-based interaction. The task manager operates in two distinct modes that have unique purposes. Each mode utilizes the same layout, icons, and interaction strategies since the types of tasks are very similar.

The first is the planning mode, where the user can plan out an engagement for simulation. Planning can be done for both hostile and friendly troops, since modeling the engagement in its entirety is the primary goal. Adding unit and targets, planning unit paths, and setting time-on-target constraints are the primary user tasks in this mode. In live mode, the same interactions are available since the user is now interacting with a pre-generated mission plan. The user is able to view, edit, and change the engagement by altering waypoint paths, changing time-of-arrival and time-on-target information, and affect the simulation entities in real time.

The direct manipulation interactions in the live mode help compensate for the problems with precise spatial selection and manipulation in the 3D Battlespace application. Also, the use of this application grants the capability for keyboard-based input, whether using a standard keyboard or a virtual keyboard utility on a tablet PC. Furthermore, the ability for this same software package to be used in both mission planning and command and control will lower training times and costs since there is only one application for users to be trained with.

4.1 Open-source Components

Similar to the Battlespace application, the Task Manager uses a few open source libraries to speed development. For this application, a portable networking library for interfacing with Battlespace was used, as well as a platform for GUI development.

4.1.1 wxWidgets

wxWidgets is a cross-platform, open source GUI framework for building desktop applications [22]. This set of libraries uses the operating system's native widget set which means that applications built on it will adapt to whatever environment they are run in [22]. Abstracting away the mouse and keyboard devices, input is handled by mapping specific functions to be called on user-generated events such as mouse-clicks, mouse-motion, and keyboard press/releases. Since wxWidgets already contains this event-handling system, the ComponentManager and BattlespaceComponent classes are not used in the Task Manager.

Mouse clicks are handled with some care in the Task Manager – in most windowing systems the first click in the double-click action registers separately than the ensuing double-click. However, in the Task Manager, having these two interactions overlap complicated the ability for users to interact directly with the map. To solve this problem, a slight delay was added to the processing of a single-click that allows the two to be processed without overlapping. Either an interaction is registered as a single-click or as a double-click; the single-click proceeding a double-click is effectively ignored.

4.1.2 VRJuggler Portable Runtime

The VR Juggler Portable Runtime (VPR) is a component that provides a platform-abstraction layer that allows the networking, serialization, and threading to be developed in a portable manner [18]. We have developed a generic library written on top of VPR that is used in both the Battlespace application and the Task Manager. It provides the Command

class and basic networking support. This library comprises the basic objects that any application networked into Battlespace requires, allowing for quicker development of supplementary applications like the Task Manager. The use of VPR allows this same code base to be used on a variety of operating systems without requiring the code to be ported manually to deal with endian issues.

4.2 Interface Layout

When considering the overall layout of the application, special attention was paid to existing mapping applications and 2D interface design principles. The largest portion of the screen was devoted to the map, since the north-up display is the primary source of information for the user, as well as the direct-manipulation interface to the units. Below the map display is a timeline display, that can be scrolled horizontally to view the planned times for units to arrive at waypoints as well as time-on-target information. The top and right side were used for the placement of application controls, with pull-down menus and a toolbar across the top. At the very bottom of the application is a status bar that is used to provide some information to the user as to any background processes the application may be carrying out, as well as providing confirmation of unit information on selection/addition. A picture of the application can be seen in Figure 15 below.

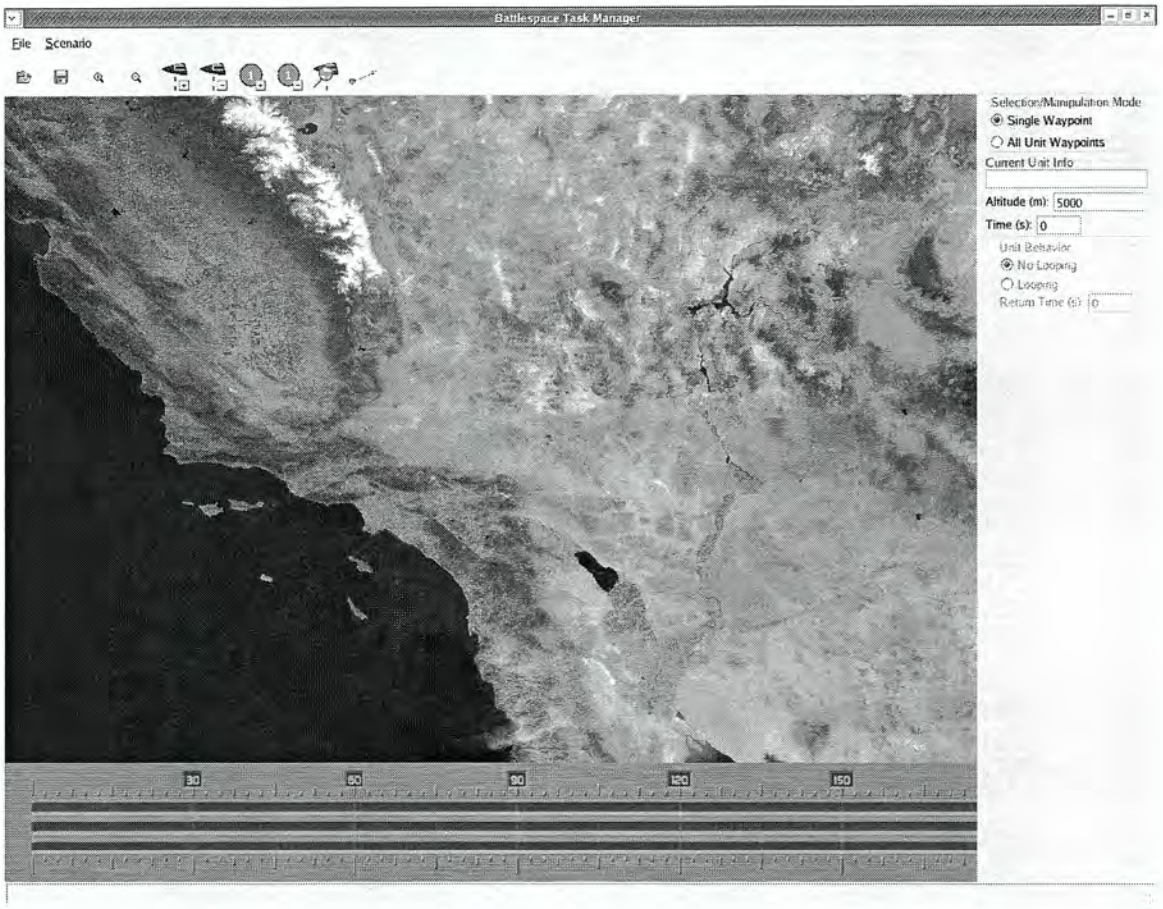

Figure 15. Task Manager

4.2.1 Map Display

The map display shows a birds-eye, north-up view of a single-resolution satellite image of the southwest United States. The eye-height above sea level can be adjusted by the user, allowing them to zoom in and out and allow for more precise interactions. Another image containing political boundaries, cities, and some information detailing roads can be overlaid on top of the satellite imagery. The positions of a unit's waypoints can be shown on top of the terrain, and are connected by a line to indicate their path. The waypoint information for multiple units is displayed all at once on a single map. The path segments between waypoints will not simply be represented by a single-pixel line; a polygon, textured

to match the unit's waypoints will be used. The reasoning behind this is that since waypoints have time-of-arrival constraints, the segments between must have speed constraints. The goal is to make these segments large enough to be selectable, such that an operator can input speed, and to add sensor-positioning controls to the segments. The icons used to display waypoint positioning are shown in Figure 16.

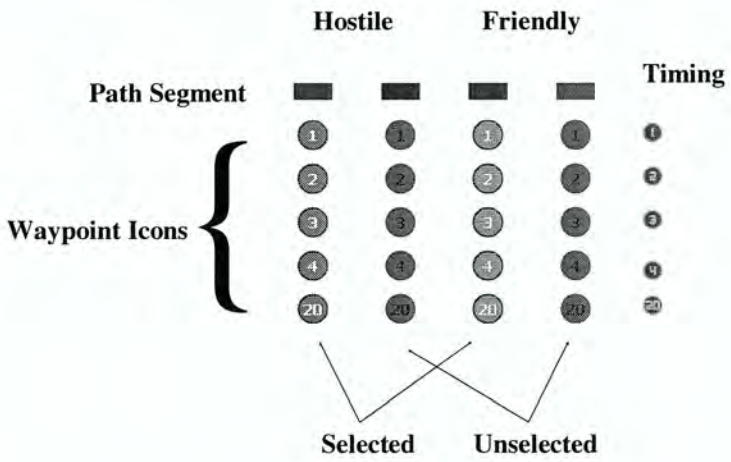

Figure 16. Waypoint Position & Timing Icons

Additionally, unit positions from a radar track or GPS can be displayed on top of this path information to indicate progress towards mission goals. It is important to note that one element that has not been included thus far is the track or link quality to that unit. In real engagements, the quality of the information coming from the unit or from the radar source is an important piece of information for the operator, but it would require additional work beyond the scope of the interface to simulate this behavior. As such, this interface assumes a high-fidelity connection between the operator's interface and the incoming signal feeds.

Displaying this information is a relevant interface challenge, but since the existing simulations do not carry this information, time was not spent specifically examining these elements. The icons used to display this positional track information of units correspond to the standard military symbols for interceptors (V) and bombers (U) and are shown below in Figure 17.

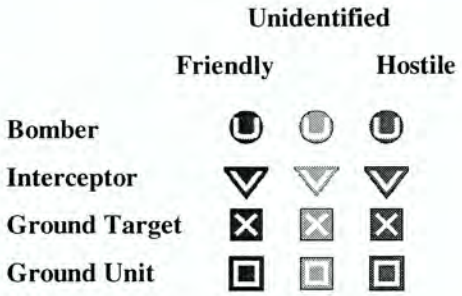

Figure 17. Unit Position Icons

4.2.2 Timeline Display

Below the map display is a timeline feed that shows projected waypoint arrival times for the currently selected unit. The units of time are given in seconds, and along the timeline every 30 seconds are labeled. Demarcation of timestamps at the lowest level is 2.5 seconds, though these marks are not labeled. Modifications of unit waypoints in the map display are automatically taken into account; the effect that changes in distance between waypoints, adding waypoints, and removing waypoints have on the arrival times are all recalculated on the fly and displayed in the timeline display.

Time on target information is also shown on the timeline display and is indicated by a red box. Multiple 'tracks' appear in the timeline display, allowing multiple units to be selected to coordinate arrival and firing times on either the same target or different ones.

4.2.3 Control Panel

The control panel has a few specific uses for the display and input of information. When a unit has been selected, the unit's type and identification is displayed on the right hand side. The altitude of that unit at a selected waypoint, and the projected time of arrival are also displayed in this panel. However, these displays hold two purposes, the displays for altitude and time-of-arrival can be overwritten by the user by clicking inside the box and typing in a new value. Additionally, spinner controls are set beside these two boxes that indicate their status as inputs as well as provide a keyless method to adjust these values.

Another set of tools exist within the right-hand control panel. When adjusting a unit's path, the user may desire to manipulate more than one waypoint at a time. If the user needs to manipulate the whole path of the unit, a simple toggle can switch between single-waypoint manipulation and all-waypoint manipulation. This allows an operator to select and manipulate all the waypoints with a single click.

The final set of controls on this panel deal with the ability to program a unit to perform a looping pattern, cycling through and repeating its path until the operator issues a different command. Here, the operator can toggle this behavior on the unit, as well as a display of the projected return time from the final waypoint in the unit's path back to the first point where it started. This time display functions just like the time display for a selected waypoint, with a spinner control and interactivity that allows a user to override a projected value.

4.2.4 Toolbar

Across the top of the application are a row of buttons that are shortcuts for specific actions. These buttons provide quicker access to the actions than the pull-down menu that sits at the topmost section of the application. These buttons also provide access to features

for the pen-based tablet PC interface with a larger surface area that are easier to click on. Figure 18 shows the buttons of the toolbar, and the functions that they are tied to.

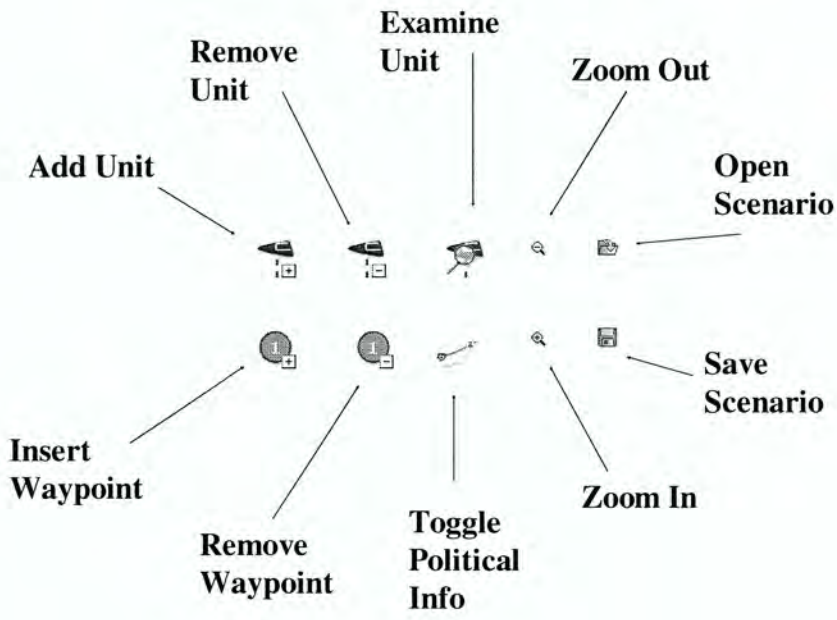

Figure 18. Toolbar Buttons

4.3 Interactions

One of the major goals of this research was to be able to combine interaction strategies such that the same interface was interoperable on two devices; a standard desktop PC with keyboard & mouse, and a tablet PC with pen-based interaction. With this in mind, multiple interactions were created for each task that a user would have. It provides the user several options, and the choice of interaction is made by the user, depending on which device their role requires them to use. The first interaction to look at is the most basic, as it's nearly

a prerequisite task for every other task a user will need to accomplish. This task is navigating the map display to a precise area with a specific field-of-view.

4.3.1 Navigation

Moving around the map display is a common task for users of the Task Manager. As such, providing an interaction directly through their input device that correlates to the display was a chief concern. Navigation through buttons that adjust the latitude and longitude of the viewpoint would be too slow for such a common task. Further, separating the manipulation of latitude and longitude into two separate interactions would also slow the task.

First, for the navigation in the desktop version, by holding the right mouse button and subsequently moving the mouse, the user can simultaneously move the coordinates of their viewpoint. The user, when clicking the right mouse button down effectively grabs that point at the map, and their motion of the mouse adjusts that position of the map within the viewpoint, thus altering the center point of the viewing frustum. Additionally, the keypad on the keyboard can be used to adjust the viewpoint one step north, west, east, or south in accordance with a north-up orientation. The eye-height of the viewpoint can be manipulated by using the mouse wheel. Mouse-wheel up, or moving the mouse wheel away from the user, will lower the viewpoint closer towards the terrain, and mouse-wheel down will raise it.

This may seem contrary to some intuitions, that mouse-wheel down should lower your position in the world, but for the user group that this is aimed at; pilots and mission planners familiar with the standard input configurations for joysticks used in manual flight, pushing forward on the device lowers your altitude, or in this case, the altitude of your viewpoint.

When navigating with the digital touch-pen, the interaction is very similar. By pressing and holding the toggle button on the pen, it can switch the processing of the touch sensor to register a secondary mode, akin to the difference between left and right mouse

clicks. By holding this button down, and pressing on the screen, the same interaction will be carried out as the user drags the pen across the map display. The pen device though, is more limited in its input options and does not provide any capability similar to the mouse wheel. Since altitude adjustment is already a separate task from latitude/longitude adjustment, the design choice to simply add buttons to the toolbar to allow control for zooming the viewpoint in and out.

4.3.2 Adding a Unit

Now that users are able to navigate the map display, the next most common task is adding units to the engagement. When adding a unit, there are two key pieces of information relevant to the task; what type of unit to add, and what its point of origin will be. There are multiple methods to add a unit, but all rely on a menu selection to determine the type. By using the drop-down menu, or by clicking the toolbar button, users can skip immediately to the type selection menu, which pops up on top of the main application, and is shown in Figure 19.

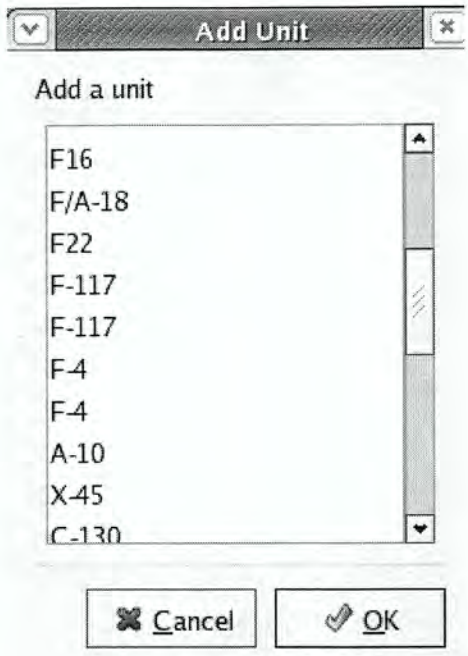

Figure 19. Add Unit Menu

Once they have chosen a type, the unit's origination point is placed on the map in the exact center of the current view. The user can then change the position of this first waypoint to a precise location. However, if the user has a specific location chosen already, but wishes to decide on which unit, the user can double-click the map display to add a unit at the location of their click. Once they have double-clicked, the position they clicked is stored, and the menu above is displayed. After their selection has been made, the icon for that unit is put into place.

4.3.3 Editing a Unit

Now that units have been placed into the scenario, the operator will want to expand upon their behavior and give them something to do. First though, to edit a unit, the user must designate which unit. Selecting a unit is as simple as single-clicking on any of a unit's waypoints or path segments. Since waypoints exist in four dimensions; the three dimensions

of physical space, as well as in time, they are visualized simultaneously on the map display, as well as on the timeline when selected.

4.3.3.1 Waypoint Positioning

One of the most common tasks is to edit the unit's path; adding, inserting, and removing waypoints to either serve as a baseline plan for a manned unit, or to be directly programmed into an unmanned unit. Adjusting a waypoint's latitude and longitude is accomplished in a manner similar to how the user navigates the map. By clicking and holding the left mouse button, the user can drag a waypoint to a new position on the map. Altitude adjustment is handled separately, through the use of an input box on the control panel that either takes keyed input or the adjacent spinner control. Only the currently selected waypoint's altitude can be adjusted in this manner.

4.3.3.2 Waypoint Timing

As waypoint positions are moved about in physical space, the timing information is automatically calculated from a default value for the speed of that particular unit based on the distances between the waypoints. However, if the user wishes to override this generated time value, they have two options for this. On the control panel, there is an input box that shows the current time-of-arrival for the currently selected waypoint, and it can be manipulated in the same manner as the altitude adjustment. Additionally, the user can click the numbered waypoints in the timeline display, and drag them back and forth horizontally to adjust times. Any input that is not a horizontal change is ignored in this input mode.

4.3.3.3 Editing Type and Targeting

The ability for a user to change their mind and edit a unit's type is critical in the planning portion. Forcing a user to recreate a path if they have decided that a particular goal of the mission would be better suited with another airframe would increase the user's

workload, force them to repeat steps, and slow down their performance. So, the unit's type can be changed at any time in the scenario creation process. All units are guaranteed to have at least one waypoint, so by double-clicking any of the unit's existing waypoints the user can bring up the edit-unit menu. This menu contains two pieces of information, the unit's unique simulation ID number, and the unit's current type. Also on this menu are two buttons. The first button, when clicked, brings up a menu similar to the add-unit menu that shows all the possible unit types. Users can review their options and select a new type, or cancel and keep the current type.

The second button on the edit-unit menu is where targeting information is accessed. Clicking on this button brings up a list of each entity in the scenario. From this list, the user can select a target and input a time that the weapon will be armed and fired. Once this has been done, a red box indicating this event will appear both on the timeline at the appropriate time, as well as on the unit's path at the position it will be at. The time can be adjusted once the firing event has been created by clicking and dragging the event along the unit's timeline. The map display's indicator will move along the unit's path as the time is adjusted, always showing the correct position for the firing event.

4.3.4 Saving and Loading Scenarios

Lastly, once an engagement has been planned to their satisfaction, or if work on the scenario must be postponed, the user can save their work for later retrieval or to be loaded into a Battlespace application. Since this is a fairly uncommon action, typically used only once per use of the Task Manager, this action can be accessed from a pull-down menu, or from a button on the toolbar. Once it has been selected, a file-browsing menu will be shown, allowing the user to select a director and filename for their scenario file. An image of this menu is shown in Figure 20.

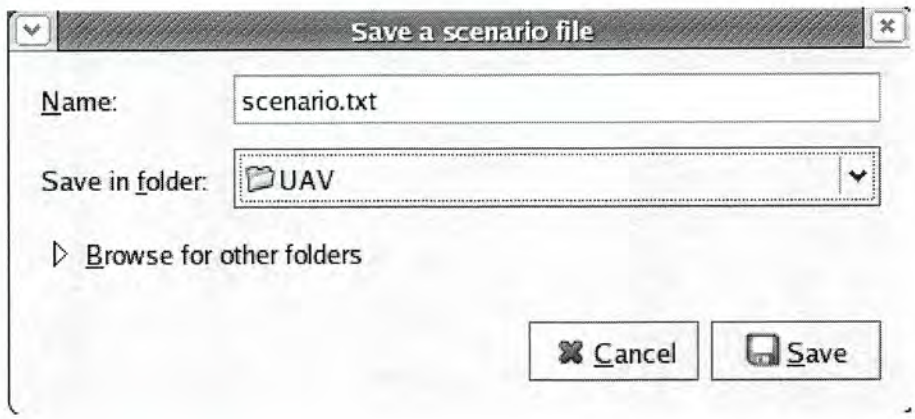

Figure 20. Save Scenario Menu

Similarly, users can load previously generated scenario files into the Task Manager, allowing work to be continued over several sessions. A special plain-text file format was created for this application, since the only applications required to load it are the Battlespace and the Task Manager.

4.4 Interaction Devices

The Task Manager has been developed to run on two different devices, both on a wireless pen-based tablet PC for use in a fully-immersive virtual reality environment, as well as on a desktop PC with keyboard/mouse interaction. The same application runs on each device, with the interaction methods for both included in a single package. This simplifies the development of the application since two versions don't have to be maintained for the different devices. The main difference in interaction between the two is the lack of keyboard input on the tablet. In addition to the use of a screen-based keyboard utility, special adaptations were made to take advantage of each device's specific features.

4.4.1 Desktop PC

In the desktop version, the availability of a keyboard allows for many actions to have quick-use shortcuts, as well as direct input of numeric information. Shortcut key actions are

available for adding and removing units, inserting and removing waypoints, as well as for bringing up the detailed unit information view. Additionally, information such as adjustments to altitude or time values can be directly input into the correct fields as precise values, rather than relying on a spinner control. Both of these interactions help increase the speed at which these actions can be completed, allowing for the operator to spend less time on manual input and more time focusing on the overall mission.

The other difference in the device is that the pointing device for the desktop, a standard two-button mouse with a center mouse wheel, provides some slight differences in how certain actions can be done. Taking advantage of the mouse wheel was a specific challenge for the design of this interface. Since, as already described above, navigation at a fixed altitude is handled by the right mouse button, it was advantageous to include the final navigation control, the altitude of the viewpoint, to the mouse. The mouse wheel provides a bi-directional control along a fixed axis, which cognitively matches the task of adjusting altitude in fixed-position display.

4.4.2 Tablet PC

The tablet PC used in the development of a wireless pen-based interaction tool is the Toshiba Tecra M4. A dual-purpose device, the Tecra M4 can be used as a conventional laptop, and it also possesses a revolving, folding screen that enables it to be used as a tablet PC. In Figure 21 below, the left hand image shows the laptop configuration, with the screen slightly rotated off-center to demonstrate this capability, and the right hand image shows the screen fully rotated and flattened into the tablet configuration with the pen device on top.

Figure 21. Toshiba Tecra M4

One of the important things for this application was to keep the screen layout consistent in both the devices; so when the Tecra M4 is in the tablet configuration, the screen stays aligned the same way, in a landscape presentation, rather than rotating into a portrait presentation, which typically the default for tablet PCs. The other main difference with the desktop version is the pointing device. The digital pen has a touch-sensitive tip that moves the cursor on the tablet, and a mode-switching button that sits underneath where the index finger grips the pen. This button is configured to switch the tip from acting as a left-button mouse click to a right-button mouse click when held. In this way, without changing the way the pen is held, two interaction styles are supported.

Since there is no mouse wheel, or comparable input device on the pen, buttons were provided for tasks that were typically completed with a mouse wheel in desktop testing. The digital pen is shown with labels in Figure 22.

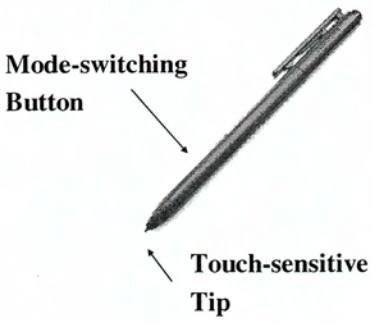

Figure 22. Touch-Sensitive Pen

CHAPTER 5. SUMMARY AND CONCLUSION

Overall, this research achieves the goals it set out to accomplish. A general framework was developed to allow rapid prototyping of interface applications that are supported by a common networking scheme. This framework promotes object independence in a way that helps developers focus in on specific tasks without compromising any flexibility. Furthermore, it supports easy extension since it only defines three base classes to accomplish this framework.

The Task Manager integrates with the existing Battlespace application using this architecture, and provides a map display similar to those used by the military in mission planning and current UAV applications. Serving dual roles, the Task Manager is used in both mission planning as well as supervisory control, giving precise control over the waypoints and paths for the unmanned units. This direct manipulation provides a level of interaction not supported with a 3D immersive application like the Battlespace. The benefits to the operator's situational awareness from the virtual environment are not compromised with a loss in interactivity.

This research holds promise as a future interface for supervisory control, whether with a single operator or with several role-defined operators with individual tasks cooperating together. Blending the benefits of multiple interaction styles and display technologies will provide the operators with the information and the tools to make effective decisions for our military forces.

5.1 Future Work

For the future state of the Task Manager, adding the capability to control the visualization in the virtual environment is a primary goal. Currently, the only way to manipulate the Battlespace visualization is using its primary input devices, the controller and voice input to directly toggle features or to manipulate the embedded menu system. Using

both the tablet PC with a pen and the controller, or even a keyboard/mouse with the controller requires too much device switching for the operator. The underlying architecture will allow for this to be accomplished in a relatively short time.

Additionally, adding the capabilities for more unit types and customizable payloads would be useful to simulate a larger array of mission types. Experimenting with planning sensor placement and orientation along a path, and allowing interactive control would allow us to determine if the roles of pilot and sensor operator can be combined with the current interface. Continued user studies will be crucial to understanding the different roles required for supervisory control.

Finally, work has begun on a multiple-touch-sensitive table display that could replace the tablet PC. Since it is multi-touch sensing, several users can interact with the same device and effect changes in the environment. Also, unlike the tablet, it is not carried by the user and would lower arm fatigue which is a problem in prolonged use.

BIBLIOGRAPHY

[1] US Department of Defense (2002). *Unmanned Aerial Vehicles Roadmap 2005-2030.* http://www.acq.osd.mil/usd/Roadmap Final2.pdf

[2] Knutzon, J., Walter, B., Sannier, A., & Oliver, J. *Command and Control in Distributed Mission Training - An Immersive Approach.* NATO Symposium on Critical Design Issues for the Human-Machine Interface, Prague, Czech Republic, May 2003.

[3] Knutzon, J., Walter, B., Sannier, A., & Oliver, J. *An Immersive Approach to Command and Control.* Journal of Battlefield Technology, Vol. 7, No. 1, pp. 37-42, March 2004.

[4] Bernard, J., Cruz-Neira, C., Oliver, J., & Sannier, A. *Command and Control Embedded Training: Visualization of the Joint Battlespace.* Final Technical Report, AFRL/IFSA Contract F30602-00-2-0622, July 2004.

[5] Walter, B., Sannier, A., Reiners, D., & Oliver. J. *UAV Swarm Control: Calculating Digital Pheromone Fields with the GPU.* I/ITSEC, Orlando, Florida, December 2005.

[6] Walter, B., Knutzon J., Sannier A., Oliver J., VR Aided Control of UAVs. 3rd AIAA Unmanned Unlimited Technical Conference, Paper AIAA 2004-6320, Chicago, 2004.

[7] Posdamer, J., Dantone, J., Gershon, N., Hamburger, T., Page, W. *Battlespace Visualization: A Grand Challenge.* Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01), San Diego, CA, October 2001.

[8] Rosenblum, L., Durbin, J., Doyle, R., Tate, D. & King, R. *Situational Awareness Using the VR Responsive Workbench.* IEEE Computer Graphics and Applications, 17(4), July/August 1997

[9] Durbin, J., Swan II, J.E., Colbert, B., Crowe, J., King, R., King, T., Scannell, C., Wartell, Z., and Welsh, T. *Battlefield Visualization on the Responsive Workbench.* Proceedings of IEEE Visualization 1998 (VIS '98)

[10] FalconView Mapping System - http://www.falconview.org/ - Accessed July 2006

[11] Tso, K.S., Tharp, G.K., Tai, A.T., Draper, M.H., Calhoun, G.L., & Ruff, H.A. *A Human Factors Testbed for Command and Control of Unmanned Air Vehicles.* 22nd Digital Avionics Systems Conference, October, 2003

[12] Ruff, H.A., Calhoun, G.L., Draper, M.H., Fontejon, J.V., & Guilfoos, B.J. *Exploring Automation Issues in Supervisory Control of Multiple UAVs.* Proceedings of the Human Performance, Situation Awareness, and Automation Technology Conference, March, 2004

[13] Dusseau, D., Negro, J.A., & Brock, C. *Designing User Friendly Situational Awareness Products.* 20th Digital Avionics Systems Conference, October, 2001

[14] Richer, J., & Drury, J. *A Video Game-Based Framework for Analyzing Human-Robot Interaction: Characterizing Interface Design in Real-Time Interactive Multimedia Applications.* Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, Salt Lake City, Utah, 2006.

[15] Lemon, O., Bracy, A., Gruenstein, A., & Peters, S. *The WITAS Multi-Modal Dialogue System I.* 7th European Conference on Speech Communication and Technology (EuroSpeech), 2001.

[16] Lewis, M., Polvichai, J., Sycara, K., & Scerri, P. *Scaling-up Human Control for Large UAV Teams.* The Human Factors of Remotely Piloted Vehicles, N. Cooke (Ed.), New York: Elsevier. 2006.

[17] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[18] VRJuggler Virtual Reality Toolkit, Version 2.0.1 - http://www.vrjuggler.org/ - Accessed July 2006

[19] OpenSG Open Source Portable Scenegraph, Version 1.4 - http://www.opensg.org/ - Accessed July 2006

[20] Demeter Terrain Engine, Version 3.21 - http://www.tbgsoftware.com/ - Accessed July 2006

[21] Knutzon, J. *Managing Multiple UAV's from a 3D Virtual Environment.* Ph.D. diss., Iowa State University, 2006.

[22] wxWidgets GUI API, Version 2.6.2 - http://www.wxwidgets.org/ - Accessed July 2006

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank those individuals who have helped me during my pursuit of my Master's degree. First, I must thank my major professor Dr. Adrian Sannier for his contagious enthusiasm and inspiring persona, as well as for the lessons he has taught me. Dr. James Oliver also deserves special recognition for his role in my education this last year. Both Dr. Sannier and Dr. Oliver have had a profound impact on the way I approach my research and I thank them for their support. I would also like to thank both Dr. Eliot Winer and Dr. Derrick Parkhurst for their contributions to this research, and I offer further thanks to Dr. Winer for participating in my committee. Additionally, I would like to thank my peer mentors, Dr. Bryan Walter and Dr. Jared Knutzon who I worked alongside for the past few years. Finally, I would like to thank my wife, Sarah, for her support and patience with me throughout the years.